LOW-LATENCY LIST DECODING OF POLAR CODES WITH DOUBLE THRESHOLDING

YouZhe Fan⋆, Ji Chen⋆, ChenYang Xia⋆, Chi-ying Tsui⋆, Jie Jin†, Hui Shen†, and Bin Li†

⋆ Department of Electronic and Computer Engineering, the HKUST, Hong Kong
†Communications Technology Research Lab., Huawei Technologies, P. R. China

ABSTRACT

For polar codes with short-to-medium code length, list successive cancellation decoding is used to achieve a good error-correcting performance. However, list pruning in the current list decoding is based on the sorting strategy and its timing complexity is high. This results in a long decoding latency for large list size. In this work, aiming at a low-latency list decoding implementation, a double thresholding algorithm is proposed for a fast list pruning. As a result, with a negligible performance degradation, the list pruning delay is greatly reduced. Based on the double thresholding, a low-latency list decoding architecture is proposed and implemented using a UMC 90nm CMOS technology. Synthesis results show that, even for a large list size of 16, the proposed low-latency architecture achieves a decoding throughput of 220 Mbps at a frequency of 641 MHz.

Index Terms— Polar codes, list decoding, successive cancellation decoding, low latency, VLSI implementation

1. INTRODUCTION

Successive cancellation decoding (SCD) is proposed in [1] for decoding polar codes, and its hardware implementation is extensively studied in [2]-[12]. However, for polar codes with short-to-medium code length, the error-correcting performance of the SCD is unsatisfactory. To improve the performance, SCDs with multiple codeword candidates are proposed. They are the list decoding [13], [14] and its variants [15]-[17]. For a better performance, cyclic redundancy check (CRC) code is serially concatenated with polar codes and the CRC bits are used to choose the valid codeword from the list candidates [13], [18], [19]. As a result, the list decoding of polar codes achieves or even exceeds the performance of Turbo codes [20] and LDPC codes [13]. However, this performance improvement is at the cost of a larger list size (e.g., 16 or 32) and the increased complexity highly desires an efficient list decoding architecture. In this work, the efficient and low-latency implementation of the list decoding is explored, aiming at promoting polar codes as a competitive coding candidate in both error-correcting and implementation aspects.

The first list decoding architecture for polar codes is proposed in [21]. In [22], the pre-computation look-ahead technique [6] is used in the list decoding for a lower latency, while its memory size is tripled. In [21] and [22], a small list size of 4 and 2 are used, respectively. When the list decoding decodes an information bit, the number of the codeword candidates are doubled. To maintain a reasonable decoding complexity, once the candidate size exceeds the specific list size $\mathcal{L}$, some of the codeword candidates have to be pruned. The common pruning strategy is to sort the codeword candidates based on their metrics and keep the $\mathcal{L}$ best of them. However, the sorting operation incurs a large hardware and timing complexity, especially when $\mathcal{L}$ is large. In [23], a list decoding architecture with list size of 8 is proposed, and a Bitonic sorting network is customized for efficient sorting. Nevertheless, up to three pipeline stages are used by the sorting architecture. As a result, to implement the list decoding with large list size in hardware, list pruning architecture is critical, especially to achieve a low decoding latency.

In this work, the list pruning architecture is optimized in both algorithmic and architectural levels. Recently, instead using log-likelihood (LL) to capture the metric of the list candidates, the LL ratio (LLR) representation is used for the list decoding [24]-[25]. Benefiting from the numerical accuracy and stability of the LLR, a small and regular architecture of the memory and processing element (PE) can be used for the list decoding [24]. Therefore, in this work, LLR is used in the design of the low-latency pruning architecture of the list decoder. Very recently in [26]-[28], borrowing some advanced techniques used in the SCD implementation [2], the special constituent codes of polar codes are utilized to reduce the latency of the list decoding. However, conventional sorting strategies are still used for their list pruning and this limits the latency reduction.

2. RELATION TO PRIOR WORK

The main contributions of this work are outlined as follows:

1. Different from the previous works on list decoding [13]-[28], a double thresholding strategy (DTS) is proposed to replace the sorting strategy for list pruning.
2. In the architectural level, the architectures for DTS and threshold value update are proposed. As a result, even for a large list size, the logic delay of list pruning is very small.
3. A low-latency list decoding architecture for a large list size, i.e. 16, is implemented in the UMC 90nm CMOS technology. Its decoding latency is even smaller than that of list size of 8 [23].

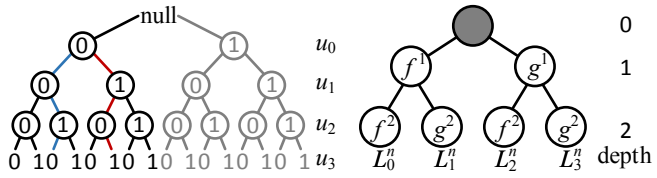
3. LIST DECODING OF POLAR CODES

A length $N = 2^n$ polar code with rate $R = K/N$ is specified by the generator matrix $\mathbf{G}_N$ and a frozen set $\mathcal{A}^c \subset \{0, 1, \ldots, N-1\}$ of cardinality $|\mathcal{A}^c| = N - K$. A source word of polar codes is denoted as $\mathbf{u}_N$, and $\mathbf{u}_N \in \{0,1\}^N$. It consists of K information bits u_i ($i \notin \mathcal{A}^c$) and $N - K$ frozen bits u_i ($i \in \mathcal{A}^c$). The information bit is used to deliver the data, while the frozen bit is set to a value, e.g., 0, pre-known by the decoder. If the r-bit CRC is used, the last r information bits take the CRC of the previous $K - r$ bits. In the encoder, the codeword $\mathbf{x}_N \in \{0,1\}^N$ is generated as $\mathbf{x}_N^T = \mathbf{u}_N^T \mathbf{G}_N$ and sent over the physical channels.

Let $\mathbf{y}$ be the noise corrupted signal of $\mathbf{x}_N$ at the receiver. The LLRs input to the decoder are given as

$$L_i^0 = \log\left[\Pr\left(\mathbf{y}|x_i = 0\right)\right] - \log\left[\Pr\left(\mathbf{y}|x_i = 1\right)\right] \qquad (1)$$

for $i = 0, 1, \ldots, N-1$. The decoding process of polar codes can be illustrated by two trees: the decoding tree and the scheduling



(a) Decoding tree and decoding path (b) Scheduling tree for SCD

Fig. 1: List decoding example of polar codes with $N = 4$ and $\mathcal{L} = 2$

tree. Fig. 1 shows an example of the trees for $N = 4$. The decoding tree of a length-N polar code is a depth-N binary tree, with u_i mapped to the nodes at depth $i + 1$. Its root node represents a null state. A path from the root node to a depth-i node represents a sub-vector $[u_0, u_1, \ldots, u_{i-1}]$ of the source word $\mathbf{u}_N$, and is named as the decoding path p^i. Specifically, a path from the root node to the leaf node of the decoding tree represents a source word $\mathbf{u}_N$ of polar codes, and the value of each bit of $\mathbf{u}_N$ is shown in the corresponding node lying at this decoding path. Notice that, if u_i is a frozen bit, it only assumes 0. Hence, the right sub-tree rooted at the depth-$(i + 1)$ node can be pruned, as the source words included in it are not valid. For example, if $\mathcal{A}^c = \{0\}$, the gray sub-tree in Fig. 1(a) is pruned.

Decoding the polar codes can be treated as a search problem in the pruned decoding tree. The conventional SCD performs a depth-first search. Given a partial decoding path $[u_0, u_1, \ldots, u_{i-1}]$, the SCD generates the LLR of bit u_i, denoted as L_i^n. If $i \in \mathcal{A}^c$, u_i is decoded as $\hat{u}_i = 0$ irrespective of L_i^n. Otherwise, a ML decision is made for the information bit u_i ($i \notin \mathcal{A}^c$), and is given by

$$\hat{u}_i = \Theta\left(L_i^n\right) = \begin{cases} 0 & L_i^n \geq 0 \\ 1 & L_i^n < 0 \end{cases} \tag{2}$$

Based on the decision rule in (2), single decoding path from the root to the leaf is obtained in the decoding tree, e.g., the red path in Fig. 1(a), and it is the source word $\hat{\mathbf{u}}_N$ decoded by the SCD.

For a better error-correcting performance, a breadth-first search is performed by the list decoding. To constrain the searching complexity, a list size $\mathcal{L}$ is set. Let $\mathcal{L}$ decoding paths at depth i of the decoding tree be denoted as $p_l^i = \left[u_0^l, u_1^l, \ldots, u_{i-1}^l\right]$, $l = 0, 1, \ldots, \mathcal{L} - 1$. For each path candidate p_l^i, a path metric is associated with it and denoted as pm_l^i. When decoding the information bit u_i, $\mathcal{L}$ decoding paths are extended to $2\mathcal{L}$ paths. From [24], the path metrics of the two extensions of the path p_l^i are given by

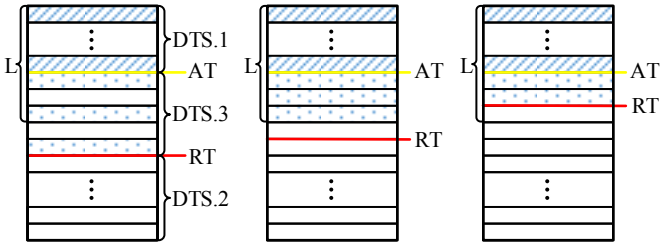
$$pm_l^{i+1}\left(u_i\right) = pm_l^i + \log\left[1 + e^{(2u_i - 1)L_i^n}\right] \tag{3}$$

where u_i assumes 0 and 1, corresponding to the left and right extensions of p_l^i. In the hardware [24], (3) is approximated by

$$pm_l^{i+1}\left(u_i\right) = \begin{cases} pm_l^i & \text{if } u_i = \Theta\left(L_i^n\right) \\ pm_l^i + \left|L_i^n\right| & \text{if } u_i \neq \Theta\left(L_i^n\right) \end{cases} \tag{4}$$

The operation in (4) is denoted as path metric update (PMU). Based on the $2\mathcal{L}$ pms from PMU, $\mathcal{L}$ extended paths with the smallest pms are chosen and they are the paths at depth $i+1$, i.e., $p_l^{i+1}, 0 \leq l < \mathcal{L}$. This operation is named as the list pruning operation (LPO).

From (3) or (4), it can be seen that the PMU needs the knowledge of L_i^n and it is generated by the SCD. The SCD operation can be described by the scheduling tree shown in Fig. 1(b). The scheduling tree of a length-N polar code is a depth-n balanced binary tree. It consists of two kinds of nodes: f node and g node. The functions included in one node can be evaluated in one clock cycle. Generally,



(a) $RT = pm_{\mathcal{L}-1}^i$ (b) A tighter RT (c) Too tight RT

Fig. 2: Double thresholding strategy

the depth-first traversal of the scheduling tree completes decoding one codeword, and the i^{th} leaf node outputs the LLR L_i^n. In the list decoding, $\mathcal{L}$ SCDs are deployed and executed in parallel. When they reach the leaf node, the SCDs are stalled and the PMU calculates $2\mathcal{L}$ path metrics from $\mathcal{L}$ L_i^ns with (4). After that, the LPO chooses the best $\mathcal{L}$ decoding paths. The SCD cannot be restarted until the LPO finishes, because the subsequent SCD operations need the knowledge of the updated decoding paths p_l^{i+1}. Therefore, the delay of the PMU and the LPO result in an increased latency of the list decoding.

From (4), the PMU is implemented with an adder array and its logic delay is small. However, a sorting strategy is used for the LPO in the conventional list decoding architecture [21]-[24]. For a shorter delay, a parallel sorting architecture [29] is used in [21] and [24]. However, its hardware complexity is $\mathcal{O}\left(\mathcal{L}^2\right)$, and hence it becomes inefficient for large $\mathcal{L}$. On the other hand, a Bitonic sorting network is used in [23], and its delay also scales with $\mathcal{L}$. Next, to achieve a short-delay LPO and hence a low-latency decoder, a double thresholding algorithm and its corresponding architecture are proposed.

4. LOW-LATENCY LIST DECODING IMPLEMENTATION

4.1. Double Thresholding Strategy

In this sub-section, a list pruning strategy with small logic delay is introduced. Based on the $2\mathcal{L}$ path metrics from PMU, it approximately finds the $\mathcal{L}$ smallest pms and their corresponding path extensions. To achieve it, the properties of the $2\mathcal{L}$ path metrics in (4) are firstly studied and presented in the following proposition.

Proposition 1. *Assume $\mathcal{L}$ path metrics at depth i of the decoding tree are sorted and*

$$pm_0^i < pm_1^i < \cdots < pm_l^i < pm_{l+1}^i < \cdots < pm_{\mathcal{L}-1}^i, \tag{5}$$

and they are extended to $2\mathcal{L}$ path metrics with (4). If the subset of $pm_l^{i+1}\left(u_i\right)$s smaller than T is defined as

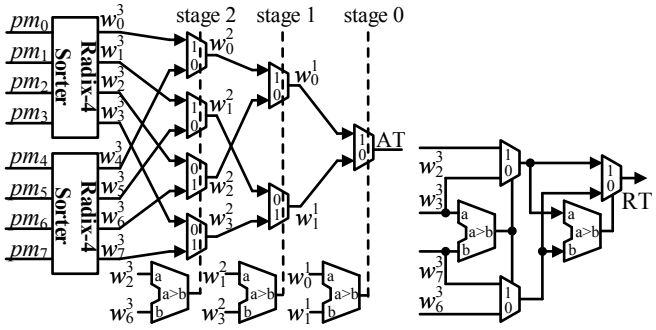
$$\Omega\left(T\right) = \left\{pm_l^{i+1}\left(u_i\right) | pm_l^{i+1}\left(u_i\right) < T\right\}, \tag{6}$$

then, the cardinality of $\Omega\left(T\right)$ for $T = pm_l^i$ satisfies

$$l \leq \left|\Omega\left(pm_l^i\right)\right| \leq 2l. \tag{7}$$

Due to the space limitation, the proof of Proposition 1 is not shown. Based on Proposition 1, the *Double Thresholding Strategy* (DTS) for list pruning is given as follows.

Double Thresholding Strategy. *Assume $\mathcal{L}$ path metrics at depth i of the decoding tree follow (5). To prune the $2\mathcal{L}$ path extensions*



(a) Architecture for AT (b) Architecture for RT

Fig. 3: Threshold tracking architecture

at depth $i+1$, two thresholds, i.e. Acceptance Threshold (AT) and Rejection Threshold (RT), are defined and set as

$$[AT, RT] = \left[pm^i_{\mathcal{L}/2}, pm^i_{\mathcal{L}-1}\right]. \tag{8}$$

The path extensions at depth $i+1$ obey the following pruning rule:

1. *if $pm_l^{i+1}(u_i) < AT$, the path extension is kept;*
2. *if $pm_l^{i+1}(u_i) > RT$, the path extension is pruned;*
3. *for path extensions with $AT \le pm_l^{i+1}(u_i) \le RT$, they are randomly chosen such that the list size remains to be $\mathcal{L}$.*

Fig. 2(a) illustrates the DTS for LPO. Assume the $2\mathcal{L}$ extended pms are sorted and the top path extension has the smallest path metric. If the list is exactly pruned, the top $\mathcal{L}$ path extensions will be the decoding paths at depth $i+1$. However, when the DTS is used, the shaded paths are reserved for depth $i+1$. As shown in Fig. 2(a), from Proposition 1, DTS.1 ensures that at least $\mathcal{L}/2$ best decoding paths are kept. Moreover, the number of the reserved paths does not exceed $\mathcal{L}$. On the other hand, since $\left|\Omega\left(pm^i_{\mathcal{L}-1}\right)\right| \ge \mathcal{L}-1$, DTS.2 efficiently excludes the path extensions that are definitely not in the set of the $\mathcal{L}$ best paths. Finally, when the number of the paths kept by DTS.1 is smaller than $\mathcal{L}$, DTS.3 will fill up the $\mathcal{L}$ path candidates. Notice that the number of the pruned paths in DTS.2 is no greater than $\mathcal{L}$. Therefore, DTS.3 is always used to fill up the decoding list.

From Fig. 2(a), the performance degradation of the DTS is due to DTS.3. If RT is loose, some decoding path belongs to the $\mathcal{L}$ best paths may not be chosen by DTS.3. To alleviate this, a tighter (smaller) RT can be assumed. For example, the value of RT in (8) can be replaced by pm^i_k ($k < \mathcal{L}-1$). As shown in Fig. 2(b), by doing so, the number of the candidates that DTS.3 can choose decreases, and hence the probability that the chosen decoding path belongs to the $\mathcal{L}$ best paths increases. However, from (7), when $RT = pm^i_k$ ($k < \mathcal{L}-1$), it is possible that more than $\mathcal{L}$ decoding paths will be pruned by DTS.2. As a result, DTS.3 is not always able to fill up the $\mathcal{L}$ path candidates, as depicted in Fig. 2(c). Hence, if RT value is too small, the performance will become poor, as the decoding paths are aggressively pruned. Therefore, an optimal value of RT exits.

Finally, from the hardware implementation perspective, the complexity of the DTS is much smaller than that of the conventional sorting strategy. To implement DTS.1 and DTS.2, $4\mathcal{L}$ comparators are sufficient and all the comparison operations can be executed in parallel as the pms are compared with the same fixed threshold values. To implement DTS.3, the circuits based on the priority encoder are used. Most importantly, due to the parallel nature of the DTS, the logic delay of the DTS is much shorter than that of the full sorting strategy. As a result, the PMU together with the DTS can be finished in one clock cycle.

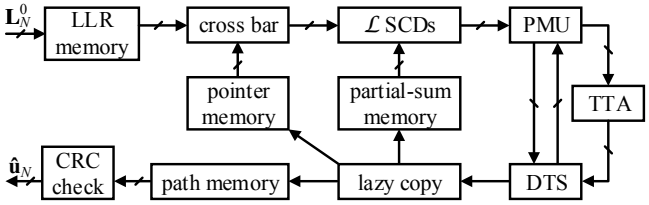


Fig. 4: Block diagram of the low-latency list decoding architecture

4.2. Threshold Tracking Architecture

To support the DTS block, the values of AT and RT are needed. These values are calculated by the *Threshold Tracking Architecture* (TTA) shown in Fig. 3. From Section 4.1, AT and RT used at depth $i+1$ of the decoding tree depend on the path metric at depth i. Therefore, the TTA can be executed in parallel with the list decoding in extending the path from depth i to $i+1$. This leads to a relaxed timing budget for TTA, and it can be executed in multiple cycles.

From (8), the TTA finds the median and the maximum values of the $\mathcal{L}$ input numbers. Finding the median is more complicated and its implementation is based on the following property of the medians.

Proposition 2. *Assume W numbers $\{w_0, w_1, \ldots, w_{W-1}\}$ satisfy the following properties:*

$$\begin{cases} w_0 \le w_1 \le \cdots \le w_{m_0} \le \cdots \le w_{W/2-1} \\ w_{W/2} \le w_{W/2+1} \le \cdots \le w_{m_1} \le \cdots \le w_{W-1} \end{cases} \tag{9}$$

where w_{m_0} and w_{m_1} are the medians of $\{w_0, \ldots, w_{W/2-1}\}$ and $\{w_{W/2}, \ldots, w_{W-1}\}$, respectively. If the median of $\{w_0, w_1, \ldots, w_{W-1}\}$ is denoted as w_m, then,

$$\begin{cases} w_m \in \{w_{m_0}, \ldots, w_{W/2-1}, w_{W/2}, \ldots, w_{m_1}\} & w_{m_0} < w_{m_1} \\ w_m \in \{w_{m_1}, \ldots, w_{W-1}, w_0, \ldots, w_{m_0}\} & w_{m_0} > w_{m_1} \\ w_m = w_{m_0} = w_{m_1} & w_{m_0} = w_{m_1} \end{cases} \tag{10}$$

Proposition 2 can be recursively used to find the value of AT. Fig. 3(a) shows the corresponding architecture for $\mathcal{L} = 8$. It consists of two radix-$\mathcal{L}/2$ sorters [29], $\mathcal{L}-1$ MUXes, and $\log_2 \mathcal{L}$ comparators. As shown in Fig. 3(a), the $\mathcal{L}$ path metrics are evenly divided into two groups and passed through the radix-$\mathcal{L}/2$ sorter. As a result, the metrics in each group are sorted, i.e., $w_0^3 \le w_1^3 \le w_2^3 \le w_3^3$ and $w_4^3 \le w_5^3 \le w_6^3 \le w_7^3$. From Proposition 2, by comparing w_2^3 and w_6^3, the size of the median candidate set is halved. In Fig. 3(a), the comparison result of w_2^3 and w_6^3 controls the 4 MUXes at stage 2 and they choose $\{w_0^2, w_1^2, w_2^2, w_3^2\}$ based on (10). Moreover, $w_0^2 \le w_1^2$ and $w_2^2 \le w_3^2$. Hence, similar comparison and MUX architectures can be used for the following stages. As a result, after $\log_2 \mathcal{L}$ stages, the median of the inputs, i.e., AT for the next depth, is obtained.

To find the value of RT, the architecture is simpler. If the maximum path metric is adopted as RT as (8), the maximum of w_3^3 and w_7^3 in Fig. 3(a) is RT. If the second maximum path metric is taken for a tighter RT, it can be found by the architecture in Fig. 3(b).

4.3. List Decoding Architecture

The top-level architecture of the proposed low-latency list decoding is shown in Fig. 4. It contains $\mathcal{L}$ SCDs and each SCD is implemented with a semi-parallel architecture of $M < N/2$ processing elements (PEs) [10]. Based on L_i^ns output from the SCD, the PMU generates $2\mathcal{L}$ pms from $\mathcal{L}$ stored pms with (4). Out of these $2\mathcal{L}$ pms, $\mathcal{L}$ pms are chosen by the DTS and they are stored in the register of the PMU.

CC	0	1	2	3	4	5	6	7	8	9	10	11
	f^1	f^2	DTS	LCP	g^2	DTS	LCP	g^1	f^2	DTS	LCP	g^2
				TTA	TTA		TTA	TTA			TTA	TTA

Fig. 5: Timing diagram of the low-latency list decoding architecture

CC	0	1	2	3	4	5	6
	f^1	PMU	g^1	f^2	DTS	LCP	g^2
			TTA	TTA		TTA	TTA

Fig. 6: List decoding timing diagram with frozen sibling

Based on registered pms, the TTA computes AT and RT used in decoding u_{i+1}. After DTS, the memory contents related to the SCD need to be copied as [21]. As shown in Fig. 4, the lazy copy (LCP) block generates the control logic for them. Finally, when the list decoding reaches the leaf node of the decoding tree, the contents of the path memory are passed to the CRC check block. The source word that satisfy the CRC check is the decoding result $\hat{\mathbf{u}}_N$.

Fig. 5 shows the timing diagram of the proposed low-latency list decoding architecture, using the example in Fig. 1 for illustration. For simplicity, assume there are already $\mathcal{L}$ decoding paths in the list in the beginning. From Fig. 5, different from the conventional SCD, two additional clock cycles are inserted after each leaf node SCD operation of the scheduling tree. As depicted in Fig. 5, they are used for the list pruning by DTS and the memory manipulation by LCP, respectively. As a result, the latency of decoding one codeword in terms of clock cycle number is given by

$$\tilde{T}_d = 4N + (n - 2 - \log_2 M)\, N/M. \tag{11}$$

Finally, Fig. 5 also shows that the TTA is not on the critical path of the list decoding. At least 2 clock cycles are available for the TTA.

4.4. Further Latency Reduction

In this sub-section, frozen siblings are used to reduce the decoding latency. They are defined as $[u_{2j}, u_{2j+1}]$ with $\{2j, 2j+1\} \subset \mathcal{A}^c$. With $0 \le j < N/2$, a frozen sibling corresponds to a leaf sibling in the scheduling tree. For a general sibling, as shown in Fig. 5, f^n and g^n are sequentially evaluated based on the LLR $\left[L_{2j}^{n-1}, L_{2j+1}^{n-1}\right]$ of their parent in the scheduling tree. However, for a frozen sibling, its path extension is fixed and given by $[\hat{u}_{2j}, \hat{u}_{2j+1}] = [0, 0]$. Moreover, the PMU from pm_l^{2j} to pm_l^{2j+2} can be simplified as

$$pm_l^{2j+2} = pm_l^{2j} + \Theta\left(L_{2j}^{n-1}\right)\left|L_{2j}^{n-1}\right| + \Theta\left(L_{2j+1}^{n-1}\right)\left|L_{2j+1}^{n-1}\right|. \tag{12}$$

It can be proven that (12) is equivalent to (4) for the frozen sibling, and 5 clock cycles can be saved by (12). For example, if $[u_0, u_1]$ in Fig.1 is a frozen sibling, the timing diagram of the list decoding is shown in Fig. 6. All the decoding operations related to the frozen sibling shrinks to a PMU operation (12) in one clock cycle. Therefore, the latency of the proposed list decoding is reduced to

$$T_d = 4N + (n - 2 - \log_2 M)\, N/M - 5FS, \tag{13}$$

where FS is the number of frozen siblings in the given polar codes.

5. EXPERIMENTAL RESULTS

An $(N, R, r) = (2048, 1/2, 16)$ polar code is sent over the BAWGN channel and decoded with different decoders. Their frame error rate (FER) curves are shown in Fig. 7. All the list decodings use (4) for PMU as in the hardware implementation. $\mathcal{L} = 2, 4, 8$, and 16

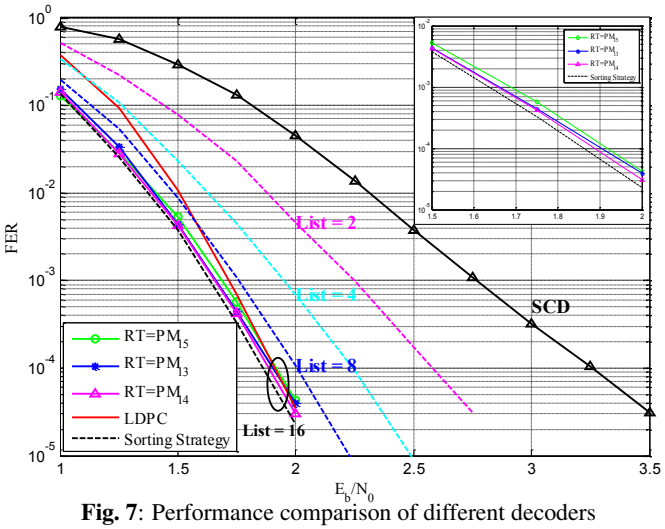


Fig. 7: Performance comparison of different decoders

Table 1: SYNTHESIS RESULTS AND COMPARISON

	This Work	[21]	[23]	[24]
Technology	90 nm CMOS			
LL/ LLR	LLR-based	LL-based	LL-based	LLR-based
N	1024			
$\mathcal{L}$	16	4	8	4
M	64			
Area (mm^2)	7.46	3.53	8.64	1.743
Clock Freq. (MHz)	641	314	625	412
Throughput ($Mbps$)	220	124	177	162

are simulated for conventional list decoding with sorting strategy [24]. As a reference, the performances of the SCD and $(N, R) = (2304, 1/2)$ WiMAX LDPC code [30] are also presented. Here, 25 iterations are used for LDPC decoding. It can be seen that the performance of polar codes is better than that of the LDPC code, when $\mathcal{L} = 16$ list decoding is used. Finally, three DTSs are used for $\mathcal{L} = 16$. Their RTs assume pm_{13}^i, pm_{14}^i, and pm_{15}^i, respectively, and AT is fixed to pm_8^i. Fig. 7 indicates that pm_{14}^i is the optimal value of RT and the performance degradation of the resulting low-latency list decoding is smaller than 0.02 dB.

The architecture shown in Fig. 4 is implemented for $\mathcal{L} = 16$ to decode $(N, R) = (1024, 1/2)$ polar codes. The quantization scheme of [24] is used, i.e., 6 bits for channel LLR L_i^0 and 8 bits for path metric. The design is synthesized using a UMC 90 nm CMOS technology, and Table I summarizes the synthesis results. Due to a large list size, the area of the LLR memory in our implementation is large and equals to 4.5 mm^2. For the target polar codes, $FS = 231$ and decoding throughput can be obtained from (13). From the table, the proposed architecture achieves a decoding throughput of 220 Mbps, and it is even greater than that of list size of 8 in [23]. The results in Table I demonstrate the effectiveness of the proposed low-latency list decoding architecture with double thresholding.

6. CONCLUSION

For a low-latency list decoding, a double thresholding strategy (DTS) is proposed for fast list pruning. With a negligible performance degradation, the DTS greatly reduces the pruning logic delay. Based on the DTS, the low-latency list decoding architecture is proposed. Comparison results demonstrate that the proposed architecture achieves a much lower latency for a large list size.

7. REFERENCES

[1] E. Arıkan, "Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inform. Theory*, vol. 55, no. 7, pp. 3051-3073, July 2009.

[2] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378-1380, Dec. 2011.

[3] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 725-728, Apr. 2013.

[4] Z. Huang, C. Diao, J. Dai, C. Duanmu, X. Wu, and M. Chen, "An improvement of modified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 17, no. 12, pp. 2360-2363, Dec. 2013.

[5] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal. Process.*, vol. 61, no. 2, pp. 289-299, Jan. 2013.

[6] C. Zhang and K. K. Parhi, "Low-Latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Trans. Signal. Process.*, vol. 61, no. 10, pp. 2429-2441, May 2013.

[7] B. Yuan and K. K. Parhi, "Low-Latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 4, pp. 1241-1254, Apr. 2014.

[8] C. Zhang and K. K. Parhi, "Latency analysis and architecture design of simplified SC polar decoders," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 2, pp. 115-119, Feb. 2014.

[9] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: algorithm and implementation," *IEEE J. Select. Areas Commun.*, vol. 32, no. 5, pp. 946-957, May 2014.

[10] Y.-Z. Fan and C.-Y. Tsui, "An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation," *IEEE Trans. Signal. Process.*, vol. 62, no. 12, pp. 3165-3179, Jun. 2014.

[11] A. J. Raymond and W. J. Gross, "A scalable successive-cancellation decoder for polar codes," *IEEE Trans. Signal. Process.*, vol. 62, no. 20, pp. 5339-5347, Oct. 2014.

[12] A. Mishra, A. J. Raymond, L. G. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. J. Gross, "A successive cancellation decoder ASIC for a 1024-bit polar code in 180 nm CMOS," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2012, pp. 205-208.

[13] I. Tal and A. Vardy, "List decoding of polar codes," 2012, arXiv:1206.0050v1 [Online]. Available: http://arxiv.org/abs/1206.0050v1

[14] K. Chen, K. Niu, and J. R. Lin, "List successive cancellation decoding of polar codes," *Electron. Lett.*, vol. 48, no. 9, pp. 500-501, Apr. 2012.

[15] K. Niu and K. Chen, "Stack decoding of polar codes," *Electron. Lett.*, vol. 48, no. 12, pp. 695-697, Jun. 2012.

[16] K. Chen, K. Niu, and J. R. Lin, "Improved successive cancellation decoding of polar codes," *IEEE Trans. Commun.*, vol. 61, no. 8, pp. 3100-3107, Aug. 2013.

[17] K. Niu, K. Chen, and J. R. Lin, "Low-Complexity sphere decoding of polar codes based on optimum path metric," *IEEE Commun. Lett.*, vol. 18, no. 2, pp. 332-335, Feb. 2014.

[18] K. Niu and K. Chen, "CRC-Aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668-1671, Oct. 2012.

[19] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 2044-2047, Dec. 2012.

[20] K. Niu, K. Chen, and J. R. Lin, "Beyond Turbo codes: rate-compatible punctured polar codes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 3423-3427.

[21] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 8, pp. 609-613, Aug. 2014.

[22] C. Zhang, X. You, and J. Sha, "Hardware architecture for list successive cancellation polar decoder," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 209-212.

[23] J. Lin and Z. Yan, "Efficient list decoder architecture for polar codes," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 1022-1025.

[24] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-Based successive cancellation list decoding of polar codes," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2014, pp. 3903-3907.

[25] B. Yuan and K. K. Parhi, "Successive cancellation list polar decoder using log-likelihood ratios," presented in *Asilomar Conf.*, 2014, arXiv:1411.7282 [Online]. Available: http://arxiv.org/abs/1411.7282

[26] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes," 2014, arXiv:1405.4819v1 [Online]. Available: http://arxiv.org/abs/1405.4819v1

[27] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Increasing the speed of polar list decoders," 2014, arXiv:1407.2921v1 [Online]. Available: http://arxiv.org/abs/1407.2921v1

[28] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Trans. Very Large Scale Integr. Syst.*, to appear.

[29] L. Amaru, M. Martina, and G. Masera, "High speed architectures for finding the first two maximum/minimum values" *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 12, pp. 2342-2346, Dec. 2012.

[30] *Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, IEEE 802.16e, Oct. 2005 [Online]. Available: http://www.ieee802.org/16/tge